AD

AD-E403 608

Technical Report ARWSE-TR-14011

ALLAN VARIANCE CALCULATION FOR NONUNIFORMLY SPACED INPUT DATA

Naomi Zirkind

January 2015

U.S. ARMY ARMAMENT RESEARCH, DEVELOPMENT AND ENGINEERING CENTER

Weapons and Software Engineering Center

Picatinny Arsenal, New Jersey

UNCLASSIFIED

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-01-0188*

1. REPORT DATE (DD-MM-YYYY) January 2015	2. REPORT TYPE Final	3. DATES COVERED (*From - To*)
4. TITLE AND SUBTITLE ALLAN VARIANCE CALCULATION FOR NONUNIFORMLY SPACED INPUT DATA		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHORS Naomi Zirkind		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, WSEC Tactical Effects, Protection & Interactive Technologies Directorate (RDAR-WSH-N) Picatinny Arsenal, NJ 07806-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army ARDEC, ESIC Knowledge & Process Management (RDAR-EIK) Picatinny Arsenal, NJ 07806-5000		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) Technical Report ARWSE-TR-14011
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

The Allan Variance (AV) characterizes the temporal randomness in sensor output data streams at various times scales. The conventional formula for calculating the AV assumes that the data samples are evenly spaced in time. The problem addressed here is that in some data streams, the data samples are not evenly spaced in time. This report presents a modified approach to AV calculation, which accommodates nonuniformly spaced time samples. The basic concept of the modified approach is that all bins used for averaging have the same time duration, though they may have different numbers of samples. The report presents and documents the Matlab program that was used to implement the modified approach. A key advantage of the approach presented here is that it can accommodate input data streams with arbitrary time spacing between data points. Also, it automatically calculates the range of valid arguments to the AV function, and it calculates and plots the AV only for arguments in that range.

15. SUBJECT TERMS

Allan deviation Nonuniformly spaced time samples Gyroscope Noisy data

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Naomi Zirkind
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	SAR	35	19b. TELEPHONE NUMBER (Include area code) (443) 861-1438

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

CONTENTS

FIGURES

FIGURES
(continued)

BACKGROUND

Devices with time varying output have some randomness (i.e., uncontrolled fluctuations) in their output. Thus, for example, a sensor that measures environmental data in real time has a nonzero output even when no data is being measured. Such nonzero output is purely noise since the signal component is zero, and this noise can be analyzed in order to derive performance specifications for the device.

The Allan Variance (AV) is a function of time that is used to quantify one aspect of the device's output and that is the bias stability. To understand what bias stability is, first bias needs to be defined. Bias is a long-term average of a time varying data stream. Consider, for example, figure 1, which shows some typical output of a Crossbow Technology VG700CA (ref. 1) gyroscope (gyro) while the unit is at rest.

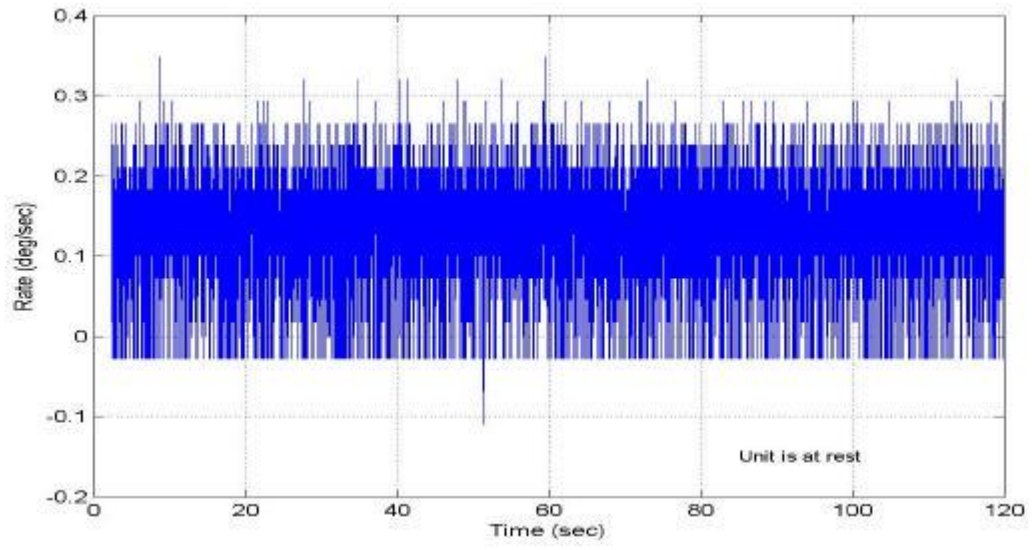


Figure 1
VG700CA rate output (ref. 1)

A quick inspection of the data in figure 1 shows that a 120-sec duration seems to be a sufficiently long duration over which to calculate a long-term average (i.e., the bias). The bias for this data is about 0.15 deg/sec. Some questions arise in connection with this calculation. Firstly, this bias value was calculated using an averaging time of 120 sec. This is not the only possible averaging time that could have been used. Furthermore, since this bias value is an average of a set of random numbers, it itself is a random number. Therefore, a number of independent samples of this bias should be taken and averaged somehow in order to derive a bias value with higher confidence.

An illustration of taking multiple averages of the gyro output using different time periods can be seen in the next few figures. Figure 2 illustrates averaging multiple 1-sec intervals, figure 3 illustrates averaging multiple 10-sec intervals, and figure 4 illustrates averaging multiple 500-sec intervals.

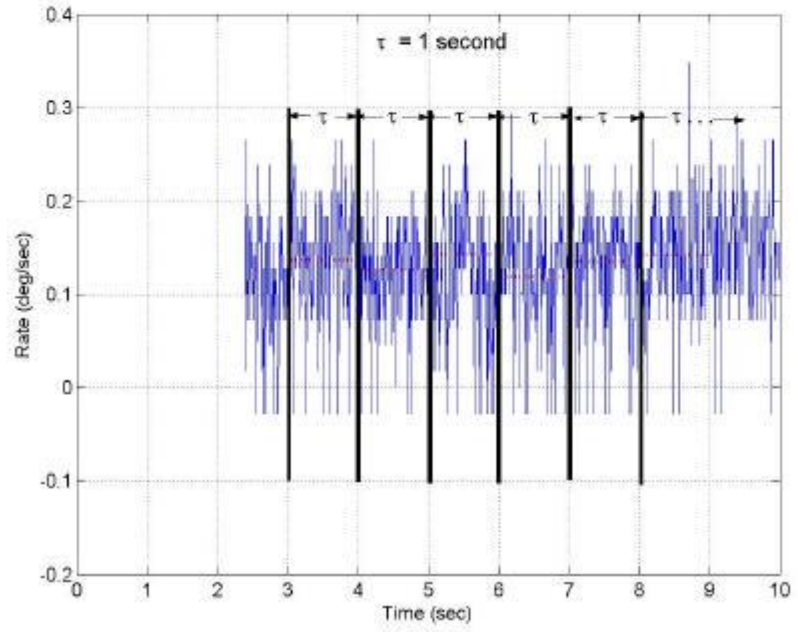


Figure 2
Data divided into bins of 1-sec length (ref. 1)

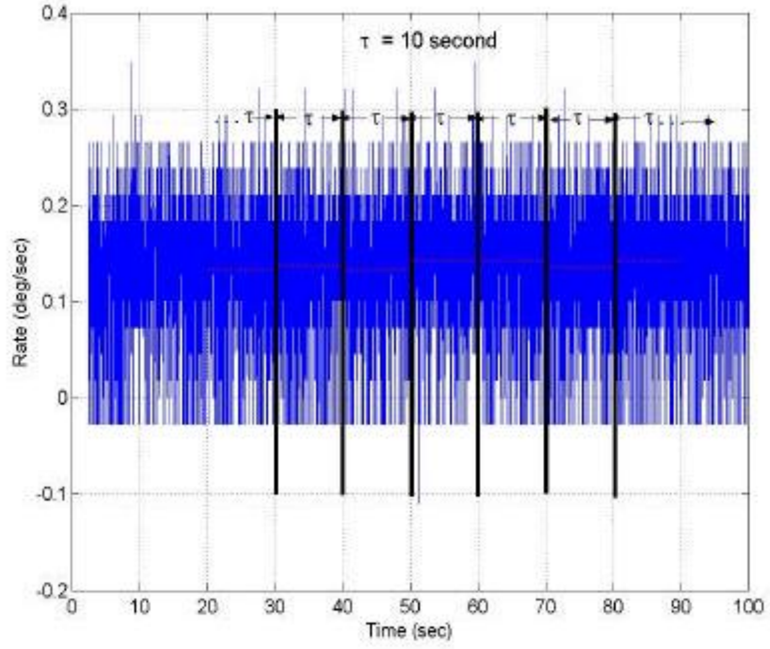


Figure 3
Data divided into bins of 10-sec length (ref. 1)

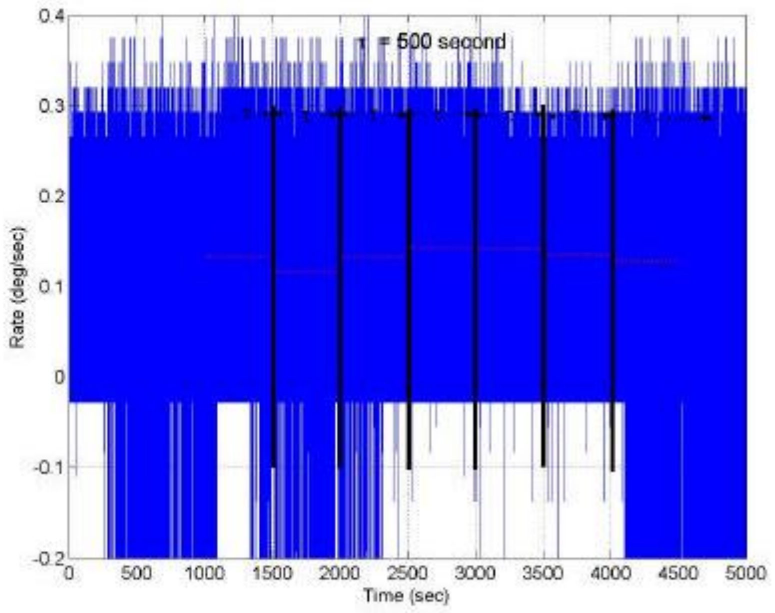


Figure 4
Data divided into bins of 500-sec length (ref. 1)

Inspection of these figures shows two things: (1) averaging over different time intervals of the same duration could give different results and (2) averaging over different durations could give different results. Thus, to thoroughly characterize the randomness, a two-fold averaging should be done.

The AV was defined to enable calculation of such averages for random, time-varying processes. The AV is a function of averaging time and shows, for each averaging time, how stable the variance of the process is over a set of independent samples. The AV has been used to characterize the randomness of the output of a variety of devices such as clocks, oscillators, amplifiers (ref. 2), and gyros.

Figure 5 shows the AV curve for the gyro whose time-dependent output is show in figure 1. At this point, it is important to clarify a point of confusion in the AV literature since this confusion is evident in figure 5. The output of a gyro is in units of angle/time (e.g., deg/sec). Thus, the variance of the gyro output is in deg^2/sec^2. However, it is more informative to present the Allan deviation (AD) rather than the AV since the AD is in the same units as the gyro output. The AD is simply the square root of the AV. The confusion arises from the fact that in verbal discussions, the AV is described, but in plots of data, the AD is presented. So, for example in figure 5, the title of the graph is correct, and the units of the y-axis are deg/hr. However, the title of the y-axis should be “Allan Deviation” rather than “Allan Variance.” In the remainder of this report, careful attention will be given to the distinction between AV and AD.

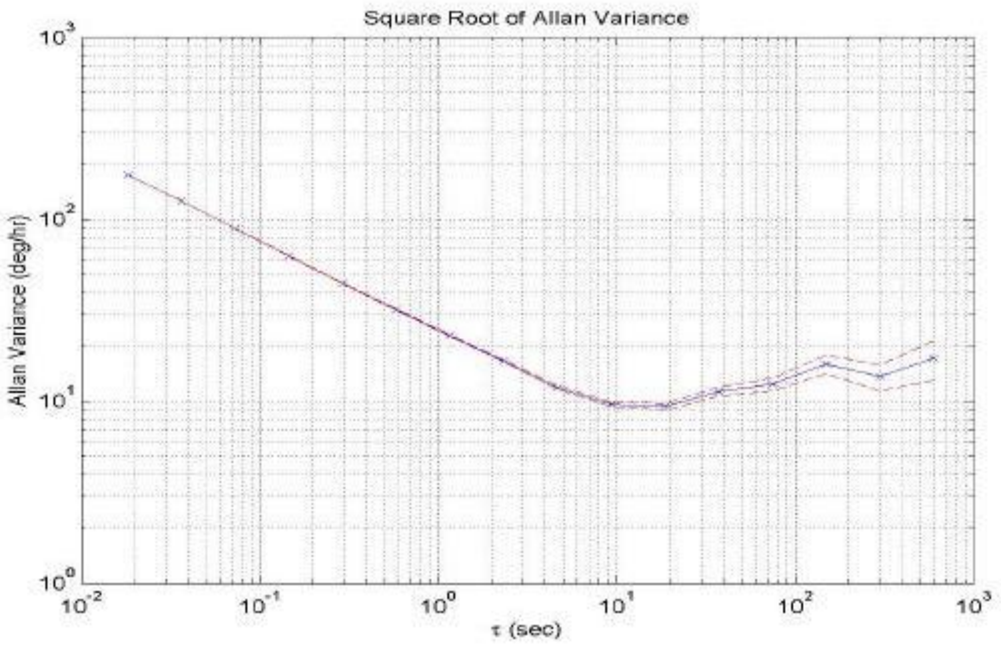


Figure 5
AD curve for Crossbow Technology VG700CA gyro (ref. 1)

This report focuses on the use of AV to characterize gyro output. Besides being informative in itself, the AD curve is also the source of some gyro specifications, i.e., bias stability and angle random walk (refs. 1 and 3).

The foregoing discussion describes the calculation of AD for uniformly spaced time samples of the gyro data. In some cases, the time samples are not uniformly spaced due to some irregularities in the data collection process. In such cases, the conventional method for AD calculation is not applicable, and a new method must be derived. One approach is to somehow interpolate “missing” data and apply second difference statistics (ref. 2) to the reconstructed time series (refs. 4 and 5, both of which are cited in ref. 6). This approach has the disadvantage that the researcher is imposing statistical features on the data that he/she is trying to characterize.

An improved technique for representing AD for nonuniformly spaced time samples is presented in reference 6. This approach uses all available data and does not do any interpolation. In the scenario described in reference 6, data was collected only on Mondays, Wednesdays, and Fridays, and thus the intervals between measurements are two days, two days, and three days. This approach uses a modified second difference operator based on the two different time intervals (two days and three days), and the AV is proportional to the expected value of the squared operator. A limitation of this approach is that, as presented, it is applicable to time series with only two different time intervals. The approach presented here is applicable to time series with time intervals of arbitrary duration that are arbitrarily distributed throughout the time series. Furthermore, the approach presented here automatically determines the minimum and maximum valid arguments for the AV function.

SUMMARY

As a baseline for the new approach to be presented, the report first presents pseudocode for the conventional approach for calculating the AV. Then, the report presents pseudocode for the modified approach, which accommodates nonuniformly spaced time samples. Then, the report presents and documents Matlab code that was used to implement the modified approach and presents a typical AD plot that was produced by the Matlab program. AD plots are presented for evenly spaced input data as well as for unevenly spaced input data showing good agreement between comparable datasets.

INTRODUCTION

This report first describes the conventional definition and method of calculation of the AV. One assumption that is made in the conventional method is that the time samples of the gyro output that are used in the calculation are evenly spaced in time. In actual systems for measurement of gyro noise, it can happen that the time samples are not uniformly spaced. In such a case, the conventional formulas are not applicable, and a modified method is required in order to calculate an AD curve. This report presents such a modified method for calculating the AV of data sets in which the samples are not uniformly spaced in time.

In particular, the report first presents the conventional formula for AV calculation and presents pseudocode for a computer program to calculate it. Then, the report presents modified pseudocode for calculating the AV in the presence of nonuniformly sampled noise data. Then, Matlab code is presented, which can be used to calculate the AV for either uniformly or nonuniformly sampled noise data. Finally, some AV curves that were calculated using the Matlab code are presented.

METHODS, ASSUMPTIONS, AND PROCEDURES

This section first describes the conventional AV equation and the concepts behind it. Then, pseudocode for calculating the conventional AV is presented. Then, pseudocode for calculating the modified AV is presented followed by a presentation of Matlab code for implementation of that pseudocode.

Conventional Allan Variance Equation

The basic concept of the conventional method of calculation of AV is as follows. The input data is a set of evenly spaced time values together with gyro output data (measured while the unit is at rest) in units of angle/time for each of the time values. The output of the calculation is an AD function in units of angle/time versus averaging interval in units of time.

Here is how to calculate the AV for a particular value of the sampling duration τ (tau). First, the set of gyro values is partitioned into bins of duration τ. For example, if the sampling duration τ is 2 sec and there are 4,000 sec of data, then there will be 2,000 bins of data. Next, the average gyro value for each bin is calculated. Then, the square of the difference between the averages for each adjacent pair of bins is calculated. Finally, the average value of all the squared differences is calculated. In equation form, the AV is defined (ref. 3) as follows

$$AV(\tau) = \frac{1}{2\,(N-1)} \sum_{i=2}^{N} \left(a(\tau)_i - a(\tau)_{i-1}\right)^2 \qquad (1)$$

where N is the number of bins of gyro data and $a(\tau)_i$ denotes the average over the ith bin of duration τ.

It is important to note that not all values of τ are valid arguments for use in equation 1. A basic rule of thumb in calculating averages as part of the AV calculation is that at least nine data values should be used in calculating the average (refs. 1 and 3). Note that in equation 1, two averages are being calculated: the average a over each bin and the average over all of the i bins. To calculate a valid average over each bin, each bin should contain at least nine values, so that

$$\tau \geq 9\,\Delta t \tag{2}$$

where Δt is the temporal spacing between gyro data values. Thus, equation 2 gives a lower limit for τ. To calculate a valid average over all of the bins, the entire data set should contain at least nine bins of duration τ, so that

$$duration \geq 9\,\tau \tag{3}$$

where *duration* is the total duration of the dataset and is equal to $N\,\Delta t$. Thus, equation 3 gives an upper limit for τ. A typical range of valid τ values is 0.1 sec to over 1,000 sec (ref. 3).

The bias stability is one commonly used performance specification for gyros, and its value is derived from the AV curve. In particular, it is the minimum value of the AD curve; recall that the AD is the square root of the AV.

Pseudocode for Conventional Allan Variance Calculation

As a baseline to use for deriving the modified AV formula, pseudocode for the conventional AV is now presented. The pseudocode takes in gyro outputs in three dimensions (x, y, and z) as a function of time, the set of time values at which the gyro outputs are measured, the time spacing between each of the data samples, and the number of τ values for which the AV is to be evaluated. The pseudocode outputs the AV functions for each of the gyro data sets evaluated at the specified number of τ values logarithmically spaced between the minimum valid τ value and the lesser of 1,000 sec and the maximum valid τ value.

The pseudocode is now presented. The names of Matlab functions are in bold font. The inputs are

- Four column vectors: *gyrox*, *gyroy*, *gyroz,* and *tvals.*
- Δt (the time spacing between the gyro output data points).
- *numtaus*, which is the number of tau values for which to calculate the AV. In this work, *numtaus* = 250 is used.

Derived quantities are

- The number of rows N in the input data vectors.
- τ_{min} and τ_{max}, which are the minimum and maximum valid values of τ (the sampling duration).
    - $\tau_{min} = 9\,\Delta t$, where Δt is spacing of *tvals*, according to equation 2.
    - $\tau_{max} = duration/9$, where $duration = N\,\Delta t$ is the total time duration of the datasets, according to equation 3.
- *tauvals*, which is a set of *numtaus* logarithmically spaced points between τ_{min} and minimum (τ_{max}, 1,000 sec).

```
$duration = N * \Delta t$                                                    (4)
for $k$ = 1 to $numtaus$                                                     (5)
    $\tau = tauvals(k)$                                                      (6)
    $numbins = \textbf{fix}(duration / \tau)$                                (7)
            %Note: fix is a Matlab function[1] that rounds a number toward zero.
    for $i$ = 1 to $numbins$                                                 (8)
        $si = (i - 1) * \textbf{fix}(\tau / \Delta t) + 1$   %Note: This is the starting index of the bin.   (9)
        $ei = i * \textbf{fix}(\tau / \Delta t)$             %Note: This is the ending index of the bin.    (10)
        $avgsx(i) = \textbf{mean}(gyrox(si{:}ei))$                           (11)
            %Note: mean is a Matlab function[2] that returns the mean of its argument.
        $avgsy(i) = \textbf{mean}(gyroy(si{:}ei))$                           (12)
        $avgsz(i) = \textbf{mean}(gyroz(si{:}ei))$                           (13)
    end $i$ loop                                                             (14)
$allanvarx(\tau) = \frac{1}{2(numbins-1)} \sum_{i=2}^{numbins} [avgsx(i) - avgsx(i-1)]^2$     (15)
$allanvary(\tau) = \frac{1}{2(numbins-1)} \sum_{i=2}^{numbins} [avgsy(i) - avgsy(i-1)]^2$     (16)
$allanvarz(\tau) = \frac{1}{2(numbins-1)} \sum_{i=2}^{numbins} [avgsz(i) - avgsz(i-1)]^2$     (17)
end $k$ loop                                                                 (18)
```

Procedure for Modified Allan Variance Calculation

For each value of τ, the conventional AV calculation partitions the gyro data sets into bins with approximately $\tau / \Delta t$ elements in them. Thus, all bins have the same number of elements in them. However, if the gyro values are not uniformly spaced, then there is no unique value of Δt. Therefore, a new way must be found to partition the gyro data sets into bins.

The basic concept behind the modified AV calculation is that all bins should cover the same time duration τ. The first major task in the procedure is to determine the starting and ending indices for each bin of the *tvals* vector. Then, the averages over the gyro value in each bin can be computed. However, each bin can now have different numbers of values, so the averages are calculated over different numbers of values.

[1] http://www.mathworks.com/help/matlab/ref/fix.html
[2] http://www.mathworks.com/help/matlab/ref/mean.html

The following sections describe the derivation of τmin and τmax, and the pseudocode for the calculation of AV for unevenly spaced time values.

Derivation of minimum and maximum values of τ

The τmin must be defined in such a way as to ensure that, regardless of which τ value is selected, each bin will have at least nine values. Figure 6 illustrates a notional example of how τmin is calculated.

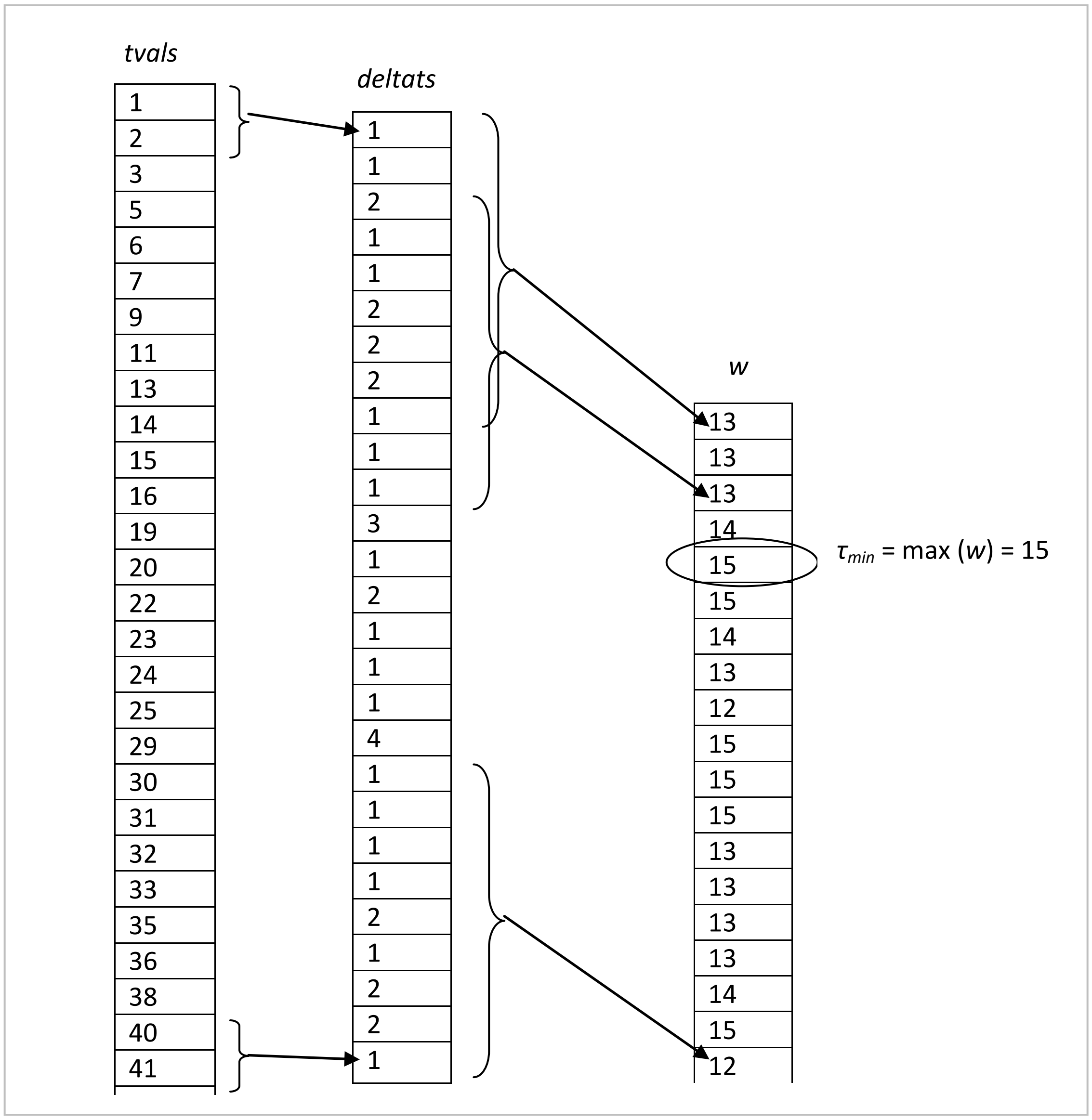


Figure 6
Procedure for calculation of τ_{min}

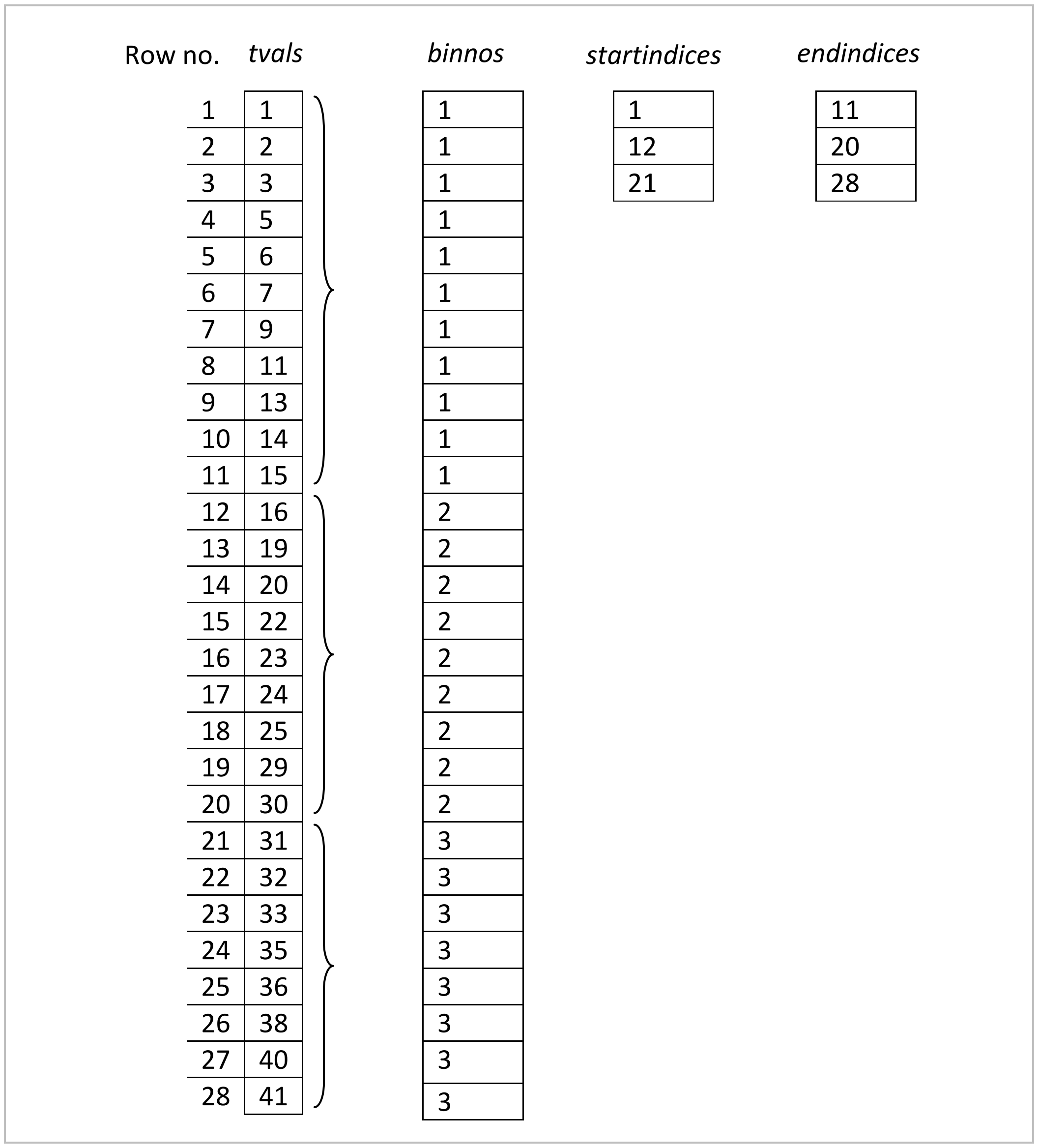
In figure 6, the left-most column represents the *tvals* vector for an example set of unevenly spaced time values in some arbitrary, unspecified units of time. The column to the right of that represents the *deltats* vector, each of whose values is the difference between adjacent values of *tvals*. An examination of the *deltats* vector clearly shows that the time values are unevenly spaced. The next step in deriving τ_{min} is to find the total time duration of each set of nine consecutive values in the *deltats* vector. This calculation gives rise to the *w* vector shown in figure 6. Next, the maximum value of *w*, which equals 15, is found. That is a special value because if τ equals 15, then each bin of *tvals* is guaranteed to have at least nine values. Of course if τ is greater than 15, each bin will certainly have at least nine values. However, if τ is less than 15, there might not be nine values in each bin. For example, consider what happens if τ equals 12. Then the first bin in *tvals* will have only eight elements. Thus, setting τ_{min} equal to 15 ensures that whatever value of τ is used, a valid average can be performed over each bin of *tvals*.

The maximum value of τ depends on the total duration of the dataset, just as in the case of uniformly spaced time values. Thus τ_{max} is equal to the total duration divided by nine. In terms of the vectors shown in figure 6, τ_{max} is the sum of the values in the *deltats* vector divided by nine.

Pseudocode for the Calculation of Allan Variance for Nonuniformly Spaced Time Values

The pseudocode for calculation of the AV for nonuniformly spaced time samples takes in gyro outputs in three dimensions (x, y, and z) as a function of time, the set of time values at which the gyro outputs are measured, and the number of τ values for which the AV is to be evaluated. The pseudocode outputs the AD functions for each of the gyro data sets evaluated at the specified number of τ values logarithmically spaced between the minimum valid τ value and the lesser of 1,000 sec and the maximum valid τ value.

One difference between this procedure and the procedure for uniformly spaced time values is the derivation of the start and end of each bin. In the procedure for uniformly spaced time values, a simple algebraic formula was used for this derivation. However, here a more complex procedure must be used. The procedure goes through the list of elements in *tvals*, dividing it into bins of temporal duration τ. Since the data points are not evenly spaced in time, the different bins will have different numbers of elements. The procedure tabulates the list of starting indices and ending indices for each bin and saves these indices for later in the procedure when averages over the bins are carried out. Figure 7 shows an example of how the starting and ending indices are assigned for the example shown in figure 6. In figure 7, the value of τ is 15.



Row no.	*tvals*	*binnos*
1	1	1
2	2	1
3	3	1
4	5	1
5	6	1
6	7	1
7	9	1
8	11	1
9	13	1
10	14	1
11	15	1
12	16	2
13	19	2
14	20	2
15	22	2
16	23	2
17	24	2
18	25	2
19	29	2
20	30	2
21	31	3
22	32	3
23	33	3
24	35	3
25	36	3
26	38	3
27	40	3
28	41	3

startindices	*endindices*
1	11
12	20
21	28

Figure 7
Calculation of starting and ending indices of bins for $\tau = 15$

Figure 7 shows that an auxiliary vector, *binnos*, is defined in order to aid in the determination of the starting and ending indices of the bins. The values of binnos are determined by dividing all of the time values in *tvals* by τ and then rounding up to the next integer. Once the *binnos* values are assigned, bin ending indices can be determined by transitions in the values of binnos, and the starting index for the next bin is simply one more than the ending index of the previous bin. Note that in figure 7, the final bin has fewer than nine elements in it, so that bin will be excluded from the averaging process that comes later in the procedure.

The pseudocode is as follows. The inputs are

- Four column vectors: *gyrox*, *gyroy*, *gyroz,* and t*vals.*
- *numtaus*, which is the number of tau values for which to calculate the AV. In this work, *numtaus* = 250 is used.

Derived quantities are

- The number of rows N in the input data vectors.
- τ_{min} and τ_{max}, which are the minimum and maximum valid values of τ (the sampling duration).
    - τ_{min} depends on the spacing of *tvals*. Slide a window over each consecutive nine element set of time values in tvals. The maximum duration from start to finish over all these windows is τ_{min} as illustrated in figure 6.
    - τ_{max} = *duration*/9, where *duration* = $N\,\Delta t$ is the total time duration of the datasets according to equation 3.
- *tauvals*, which is a set of *numtaus* logarithmically spaced points between τ_{min} and τ_{max}.
- *binnos* is the vector that tells for each value of *tvals* into which bin that value falls.
- *startindices* is the vector of initial values of the bins.
- *endindices* is the vector of final values of the bins.

```
for $k$ = 1 to $numtaus$                                                    (19)
    $\tau$ = $tauvals(k)$                                                   (20)
    Initialize $binnos(1) = startindices(1) = 1$                            (21)
    for $j$ = 2 to $N$                                                      (22)
        If $binnos(j)$ is a multiple of $\tau$,                             (23)
        Then $binnos(j)$ = integer part of $tvals(j)\ /\ \tau$              (24)
        Else $binnos(j)$ = 1 + integer part of $tvals(j)\ /\ \tau$          (25)
        Endif                                                               (26)
        If $binnos(j) > binnos(j-1)$                                        (27)
        Then we have just started a new bin                                 (28)
            Next value of endindices is $j-1$                               (29)
            Next value of startindices                                      (30)
                = 1 + new value of endindices
        Endif                                                               (31)
    end $j$ loop                                                            (32)
    Set final value of $endindices$ to $N$                                  (33)
    for $i$ = 1 to $numbins$                                                (34)
        $si = startindices(i)$                                              (35)
        $ei = endindices(i)$                                                (36)
```

```
            $avgsx(i) = \frac{1}{ei-si+1} \sum_{j=si}^{j=ei} gyrox(j)$    (37)
            $avgsy(i) = \frac{1}{ei-si+1} \sum_{j=si}^{j=ei} gyroy(j)$    (38)
            $avgsz(i) = \frac{1}{ei-si+1} \sum_{j=si}^{j=ei} gyroz(j)$    (39)
        end $i$ loop    (40)
        $allanvarx(\tau) = \frac{1}{2(numbins-1)} \sum_{j=2}^{numbins} [avgsx(i) - avgsx(i-1)]^2$    (41)
        $allanvary(\tau) = \frac{1}{2(numbins-1)} \sum_{j=2}^{numbins} [avgsy(i) - avgsy(i-1)]^2$    (42)
        $allanvarz(\tau) = \frac{1}{2(numbins-1)} \sum_{j=2}^{numbins} [avgsz(i) - avgsz(i-1)]^2$    (43)
    end $k$ loop    (44)
```

Matlab Code for Modified Allan Variance Calculation

The following section presents the actual Matlab code[3] that implements the pseudocode that was presented for calculation of AV for unevenly spaced time values. The code actually outputs the AD rather than the AV since the AD is what is usually reported in the industry as a performance specification. The code presented here is broken into sections so that each section can be explained. The entire code without breaks is presented in the appendix.

The first task is to clear all previously stored values of all variables (fig. 8).

```matlab
% Allan Variance Calculation April 2014; Written by Naomi Zirkind
% This program calculates the Allan Deviation curves for x, y, and z gyro
% data streams, and prints a plot of the Allan Deviation.
% Inputs: allanarray ([number of rows], 4)
% Derived quantities:  N (number of rows in allanarray), tvals[N],
% gyrox[N], gyroy[N], gyroz[N]
% Outputs: allandevx, allandevy, allandevz  -- all are functions of tau

clear all;
```

Figure 8
Clearing all previously stored values of all variables

Read in Data from File

The next task is to read in the array, which is assumed to be comprised of four columns – the first column is time, and the second through fourth columns are the x, y, and z values

[3] *Published with MATLAB® R2013a*

of the gyro output, respectively. Each row of the array represents the data collected at a particular time (fig. 9).

```
allanarray = xlsread('data_set_name.xlsx');
```

Figure 9
Reading in data for *allanarray*

Optional Random Removal of Rows from Allanarray to Simulate Nonuniformly Spaced Time Samples

A new array*, sparseallanarray*, is created, which is a copy of *allanarray* except that every seventh row of *allanarray* is either kept or discarded according to the value of a binary random variable *r*. After all seven-row blocks of data in *allanarray* have been processed in this manner, the number of as yet uncopied rows in *allanarray* is mod (*allanarraynumrows*, 7), where *allanarraynumrows* is the number of rows in *allanarray*. This number is used to figure out how many rows of *allanarray* still need to be copied into *sparseallanarray*.

To increase efficiency of the Matlab program, the array sparseallanarray was preallocated (i.e., filled with zeroes) before execution of the loop in which its values are assigned. Once all the data elements of sparseallanarray have been assigned, the zeroes at the end of this array that have not yet been overwritten with data from *allanarray* must be eliminated. This is done by using the Matlab “find” function, which returns the indices of the nonzero elements of its argument. The largest of these indices represents the end of data (as opposed to the zeroes) in the *sparseallanarray* matrix. The final rows of *allanarray,* if there are any left uncopied, are now copied into *sparseallanarray* beginning right after the last data element in *sparseallanarray*.

Finally, only the data values of *sparseallanarray* are copied into a new matrix, *sparseallanarray2*, which does not have the superfluous zeroes at its end. This new matrix *sparseallanarray2* is now copied into the variable *allanarray*, which is then processed according to the original Matlab code to find its AD.

This entire section is optional and self-contained, so that if the user wants to do a normal (i.e., without data removal) calculation of the AD, this entire section can simply be commented out. This new array*, sparseallanarray,* can be seen in figure 10.

```
i = 1; % index for sparse array elements
j = 1; % index for allanarray elements
allanarraynumrows = size(allanarray,1);
sparseallanarray = zeros(allanarraynumrows,4);  % preallocation

while j < allanarraynumrows - 5
   sparseallanarray(i:i+5,:) = allanarray(j:j+5,:); % copy six rows
   r = randi(2,1);   % Returns binary random variable 1 or 2 with equal probability
   if (r==2)
      sparseallanarray(i+6,:) = allanarray(j+6,:);  % Copy the seventh row
      i=i+7;
   else i=i+6; % Here, r=1. Don't copy the seventh row
   end
   j = j+7; % Move ahead to the next block of elements of allanarray
end

% copy the mod(allanarraynumrows,7)elements at the end of allanarray
remainder = mod(allanarraynumrows,7); % number of rows of allanarray than have not yet been
copied to sparseallanarray
nonzeroindices = find(sparseallanarray(:,1));  % Find nonzero elements in column 1, which is the
time values
lastnonzeroelement = nonzeroindices(end);  % last nonzero element of time values
if ~isequal(remainder,0)
   sparseallanarray(lastnonzeroelement+1:lastnonzeroelement+remainder,:) =
allanarray(allanarraynumrows-remainder+1:allanarraynumrows,:);
end

% get rid of the extra zeroes at the end of sparseallanarray
sparseallanarray2 = sparseallanarray(1:lastnonzeroelement+remainder,:);
allanarray = sparseallanarray2;  %copy the sparse allan array into allanarray for further processing
```

Figure 10
The creation of *sparseallanarray*

Break Up Big Matrix into Smaller Matrices

After the array *allanarray* is read in, it is decomposed into its individual arrays for further processing (fig. 11).

```
tvals = allanarray(: , 1);
gyrox = allanarray(: , 2);
gyroy = allanarray(: , 3);
gyroz = allanarray(: , 4);
N = length (tvals);
```

Figure 11
Allanarray broken up into individual arrays

Determine the Range of Valid τ Values

At this point in the program, the major focus is on the *tvals* array, since that data is used in order to divide the gyro data into bins for averaging for each particular τ value. The first task here is to determine the minimum and maximum valid values of τ (i.e., τ_{min} and τ_{max}, respectively). As explained earlier, τ cannot be too small or else there will not be enough elements in each bin to do proper bin averages. Furthermore, τ cannot be too large or else there will not be enough bins to average over to calculate the AD. To calculate the minimum τ value, the program checks out all windows of length nine (the minimum number of elements needed to do a proper average) and assigns a value to τ_{min} that is at least as large as the window with the largest time span. The procedure illustrated in figure 6 is followed. In assigning a value to τ_{max}, the total duration of the dataset is divided by a number just slightly larger than nine in order to ensure that nine or more bins will be available for averaging for the largest value of τ (fig. 12).

```
mask = ones(1, 9);
deltats = diff(tvals); % vector of differences of adjacent time values
w = conv(deltats, mask, 'valid');
taumin = max(w);
duration = tvals(N);
taumax = duration / 9.001;
```

Figure 12
Assigning values to τ_{min} and τ_{max}

Set up Outer Loop over τ Values

The outermost loop in the program is over the various values of τ. The values of τ to be used in the calculation range from τ_{min} to τ_{max}. The values of τ are logarithmically spaced since the AD plot is generally displayed as a log-log plot. The arrays that will eventually store the AD data are pre-allocated for improved execution efficiency. The binnos array, illustrated in figure 7, is also pre-allocated at this point. The implementation of these steps can be seen in figure 13.

```
tauvals = logspace (log10(taumin), log10(taumax), 250);
numtaus = length (tauvals);

% Preallocation
allandevx = zeros(1, numtaus);
allandevy = zeros(1, numtaus);
allandevz = zeros(1, numtaus);
binnos = zeros(1, N);
```

Figure 13
Calculation of tau values

Outer Loop over τ values

At this point begins the loop over *k*, an index for the various values of the *tauvals* array, which is the logarithmically spaced set of values between τ_{min} and τ_{max}. This loop constitutes the main body of the program. The *startindices* and *endindices* arrays whose derivation is illustrated in figure 7 are pre-allocated for two reasons: (1) in order to improve execution efficiency and (2) in order to clear all values in these arrays that were calculated in previous iterations of the *k*-loop.

Next, the *j*-loop iterates through all rows of the *tvals* array and calculates the values of *binnos*. When the loop detects an increment in *binnos*, it assigns values to the *startindices* and *endindices* arrays, which define the bins to be used for averaging.

Regardless of the value of τ, the end index for the final bin will always be N, which is the number of rows in *tvals.* Therefore, the value of *endindices* for the final bin is explicitly set to N even though it does not correspond to any increment in *binnos*. It could happen that the final bin has fewer than nine elements, and if that is the case, then that bin will be disregarded and will not be used for averaging.

The final steps in this section perform the actual AD calculation shown in equations 28 through 30. The averaging is performed over the bins whose start and end indices are the values of the arrays *startindices* and *endindices* (fig. 14).

```
for k = 1:numtaus
   tau = tauvals(k);

% Create startindices and endindices arrays based on the array of
% measurement times. Calculate endindices, and then startindices can easily
% be derived from it. binnos is an auxiliary vector whose components are
% the bin number of the corresponding element of tvals.
   startindices = zeros(1,N);   %preallocation
   endindices = zeros(1,N);     %preallocation
   binnos(1) = 1;
   startindices (1) = 1;

   % j is row number in tvals, and i is bin number.
   for j = 2:N
      if (isequal (mod (tvals(j) , tau), 0))
         binnos(j)  = uint32( tvals(j) / tau );
      else binnos(j) = uint32 ( tvals(j) / tau) + 1;
      end

      if ( binnos(j) > binnos(j-1))
         i = binnos(j-1);
         endindices(i) = j-1;
         startindices(i+1) = endindices(i)+1;
      end
   end

   endindices(binnos(N))= N;
   % If the final bin has fewer than 9 elements, exclude it from future
   % processing.
   if (endindices(binnos(N)) - startindices(binnos(N)) < 8)
      numbins = binnos(N) - 1;
   else numbins = binnos(N);
   end

% Calculate averages, deltas, squareds, sumofsquares, and allandev vectors
% Sum over each bin to get the bin averages

% Preallocation
avgsx = zeros(1, numbins);
avgsy = zeros(1, numbins);
avgsz = zeros(1, numbins);

  for i = 1:numbins
     si = startindices(i);
     ei = endindices(i);

     avgsx(i) = mean(gyrox(si:ei));
     avgsy(i) = mean(gyroy(si:ei));
     avgsz(i) = mean(gyroz(si:ei));
  end

  squaredsx = (diff(avgsx)).^2;
  squaredsy = (diff(avgsy)).^2;
  squaredsz = (diff(avgsz)).^2;

  allandevx(k) = (  sum(squaredsx) / (2 * (numbins -1))  )^0.5;
  allandevy(k) = (  sum(squaredsy) / (2 * (numbins -1))  )^0.5;
  allandevz(k)  = (  sum(squaredsz) / (2 * (numbins -1))  )^0.5;

end
```

Figure 14
Final steps performed for the actual AD calculation

Write the Allan Deviation Vectors into Excel Files

At this point, the three arrays for the AD of each gyro direction (x, y, and z) have been calculated, and they are stored as Excel files for later viewing and processing (fig. 15). The units of these vectors are assumed to be deg/sec (i.e., the same units as the input gyro data).

```
xlswrite ('tauvals.xls', tauvals);
xlswrite ('allandevx.xls', allandevx);
xlswrite ('allandevy.xls', allandevy);
xlswrite ('allandevz.xls', allandevz);
```

Figure 15
Writing the AD arrays as Excel files

Display the Vectors in a Log-log Plot

The AD data is now converted from deg/sec to deg/hr, which is the usual unit for the AD for high performance gyros. Then, the plot parameters are set up for plotting of the AD data (fig. 16).

```
allandevx = allandevx * 3600;
allandevy = allandevy * 3600;
allandevz = allandevz * 3600;

loglog (tauvals, allandevx, '-r', tauvals, allandevy, '-g',tauvals, allandevz, '-b')
grid on
xlabel ('Time (seconds)');
ylabel ('Allan Deviation (degrees / hour)');
title('Allan Deviation for Gyro Data');
legend('X-Axis', 'Y-Axis', 'Z-Axis');
```

Figure 16
Plot parameters for AD data

Display and Analysis of Allan Deviation Plots

Figure 17 shows a set of AD curves[4] calculated using the Matlab program presented here.

[4] The data come from a developmental system supplied by John Taylor of Mercury Data Systems, December 2012 and were acquired during the development process.

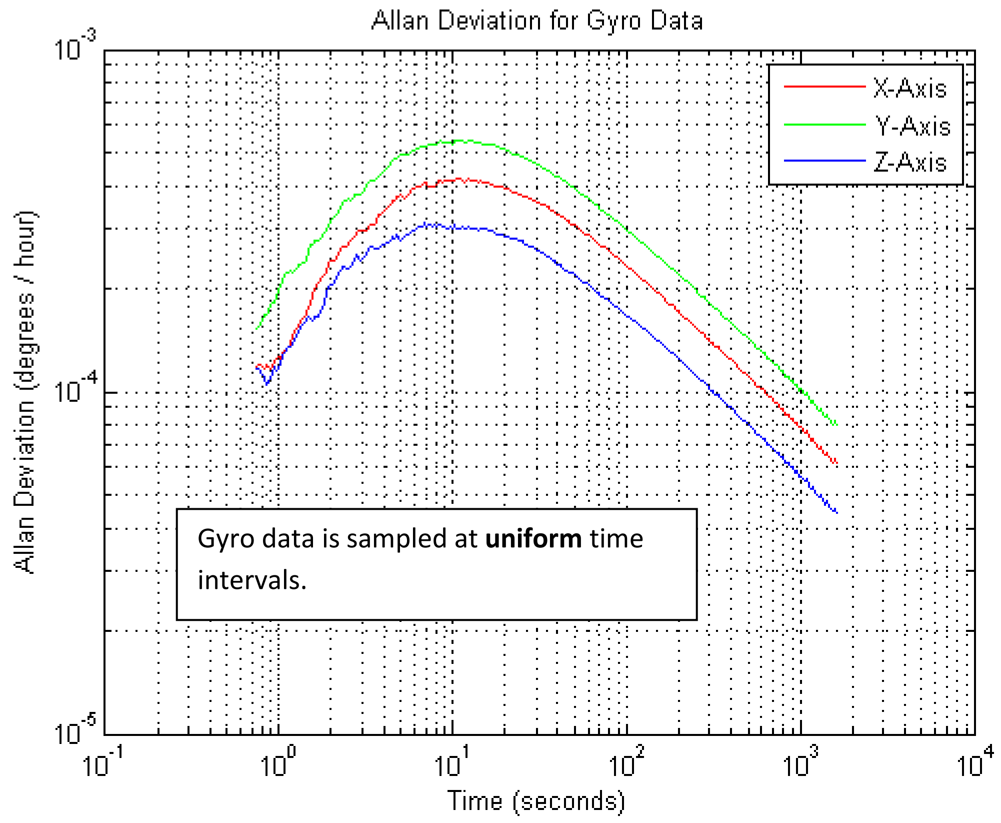


Figure 17
AD plot for gyro data of a developmental system; uniformly spaced time samples

Although the Matlab program used to generate figure 17 can accommodate input data sampled at nonuniform time intervals, the data used to calculate the curves in figure 17 actually was sampled at uniform time intervals. It is desirable to evaluate the performance of the Matlab code using data with nonuniform time intervals in order to test its full capability. An “apples-to-apples” comparison of the program’s output using a dataset with uniform samples versus its output with a comparable dataset with nonuniform samples would help validate the program’s performance.

In order to generate the comparable dataset with nonuniform samples, the dataset with uniform samples had rows of data randomly removed[5]. This data removal resulted in basically the same dataset as the original but with nonuniform samples. This random data removal represents what actually happens in some data collection sessions: sometimes data samples are corrupted or lost due to random noise process.

The data removal was performed as follows. The input dataset had uniform samples. The first six rows were copied into a new dataset. Then, a binary random number was generated that determined whether or not the seventh row would also be copied. This procedure was repeated as long as there were at least seven rows uncopied in the original dataset. The last few rows of the original dataset were then copied into the new dataset. The Matlab program was then run with this new dataset, and the result is shown in figure 18.

[5] The core concept of this suggestion was made by Thomas Bail, Air Force Nuclear Weapons Center, Kirtland Air Force Base, NM, April 2014.

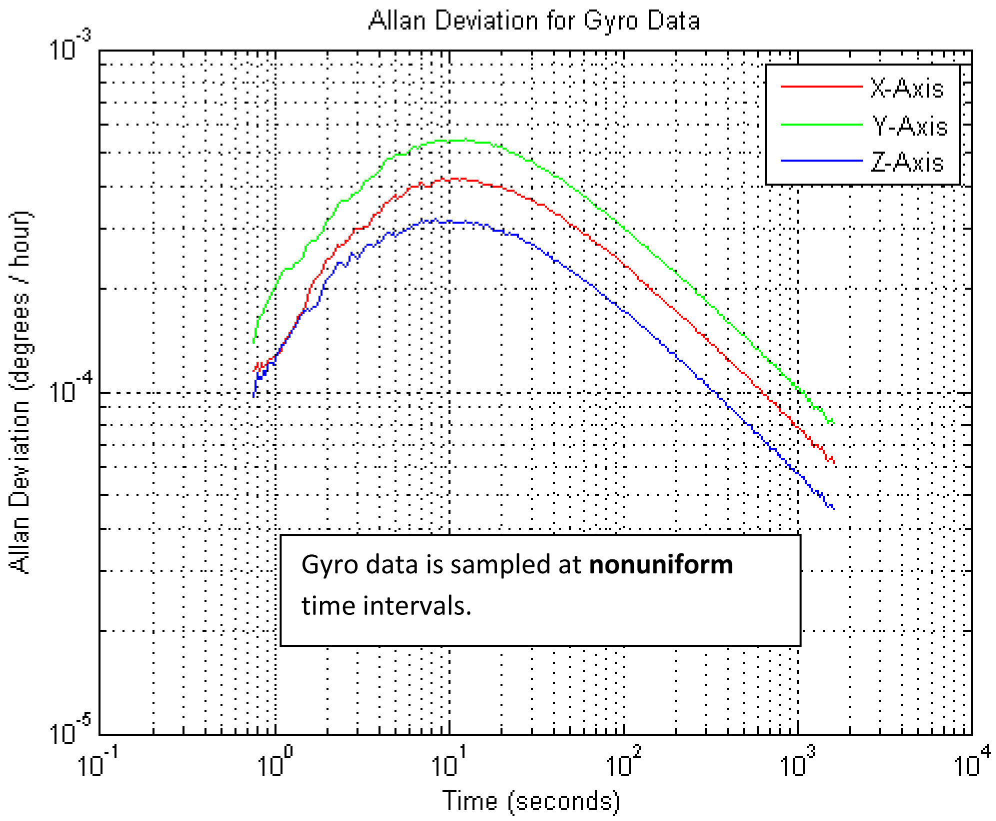


Figure 18
AD plot for gyro data of a developmental system; nonuniformly spaced time samples

A comparison of figures 17 and 18 shows that the two sets of curves are quite similar though not identical. That gives credence to the validity of the approach presented here.

As a simple check on the statistical properties of the nonuniformly spaced data, note that the original dataset had 848,682 rows. Dividing that by seven gives 121,240 seven-row blocks of data. It is expected that half of these blocks of data will have a row of data removed. Thus, it is expected that 60,620 rows of data will be removed. Actually, the dataset used to calculate the curves in figure 18 had 788,143 rows, which corresponds to an elimination of 60,539 rows - very close to the expected value of 60,620 rows.

RESULTS AND DISCUSSION

This report has presented a new technique for calculating the AV for nonuniformly spaced time samples. Pseudocode as well as Matlab code has been presented and documented with explanations for the various steps in the procedure. The Matlab code is presented in its entirety and can be copied and pasted into a Matlab editor and executed after minor changes, i.e., specification of the input file and of plot parameters.

This procedure can deal with time series with arbitrary spacing between time samples, which is a key advantage over the conventional approach. The modified approach for AV calculation that is presented here calculates the AV by binning the input data into bins of uniform time duration, although the various bins might have different numbers of data samples in them. The approach ensures that each bin has at least nine data points by restricting the values of the input argument τ to the AV curves.   It does this by automatically calculating the minimum and maximum arguments of the AV function so as to ensure statistical reliability of the result. The Matlab code plots the AV only for arguments in the statistically valid range.

A comparison was conducted by running the Matlab program with a uniformly sampled dataset and then again with a nonuniformly sampled dataset that was derived from the original by randomly removing rows of data from the original dataset. The AD curves produced for the two datasets agreed well, and the number of rows removed from the original dataset corresponded well with the expected value of how many rows would be removed. This comparison gives credence to the algorithm and Matlab code implementation that are presented here.

One caveat is connected with the fact that the AD as calculated in the Matlab program is itself a random variable since it is a function of the gyro data values that are random variables. If the AV data is calculated using an average over many samples of gyro data, the calculated AV values will have higher confidence. In the method presented here, different bins in the averaging process can possibly have different numbers of elements due to the fact that the sample values can have nonuniform time spacing between them. Thus, when viewing the AD curves, it would be prudent to consider the statistical reliability of the data points in these curves, and that consideration would be based upon examination of how many samples were used in calculating these data points.

CONCLUSIONS

The modified method for calculating Allan Variance (AV) that is presented here can accommodate input data streams with intervals of arbitrary lengths between time samples. This method enables analysis of more types of data than the conventional AV equations allow. This method can also accommodate the limiting case of uniformly spaced data samples as the conventional AV equation can do. The Matlab code has been tested with evenly spaced input data as well as with unevenly spaced input data and gives expected results, which lends credence to the algorithm and code presented here. The Matlab code presented is well documented and allows easy implementation of the presented approach.

RECOMMENDATIONS

This section describes some suggested improvements to the approach and the code presented here.

The Matlab code presented here implements the modified approach to calculation of AV for nonuniformly spaced time samples. It has been tested with a few datasets, but more exhaustive testing of the code would be desirable. Testing this code on a variety of datasets would help to validate it and/or to show where revisions are needed.

Furthermore, it would be desirable to make the Matlab code more efficient since it may have to process very large datasets. For example, the dataset that was used to generate the AD plots shown in the report has 848,683 rows of data in it. The Matlab code presented here took a few minutes to execute the code. Alternatively, the code might be more efficiently implemented in C++ or a different computer language that is better optimized for efficient execution.

An improvement to the code presented here, besides making it more efficient, is to present in the Allan deviance plot some indication of the statistical validity of the data presented in the plot. The *startinidces* and *endindices* arrays contain information on how many elements are in the bins that are used in the averaging process.

REFERENCES

1. Stockwell, Dr. W, “Bias Stability Measurement: Allan Variance,” Crossbow Technology, Inc., http://www.xbow.com/Literature/Application_Notes_Papers/Gyro_Bias_Stability_Measurement_using_Allan_Variance.pdf, accessed March 26, 2014.

2. “Allan Variance,” Wikipedia, http://en.wikipedia.org/wiki/Allan_variance, accessed March 26, 2014.

3. “Gyro Sensor Model,” VectorNav Support Library, http://www.vectornav.com/support/library?id=85 , accessed March 27, 2014.

4. Tavella, P. and Leonardi, M., “Noise Characterization of Irregularly Spaced Data,” Proceedings of the 12th European Frequency and Time Forum, Warsaw, Poland, p. 209-214, 10-12 March 1998.

5. Hackman, C. and Parker, T.E., “Noise Analysis of Unevenly Spaced Time Series Data,” Metrologia, vol. 33, p. 457-466, 1996.

6. Davis, J. A., Harris, P. M., Shemar, S. L., and Cox, M. G., “Least-Squares Analysis of Two-Way Satellite Time and Frequency Transfer Measurements,” presented at 33rd Annual Precise Time and Time Interval Meeting, Long Beach, CA, DTIC Report ADA485223, 27-29 November 2001.

APPENDIX
MATLAB CODE

```
% Allan Variance Calculation April 2014
% This program calculates the Allan Deviation curves for x, y, and z gyro
% data streams, and prints a plot of the Allan Deviation.
% Inputs: allanarray ([number of rows], 4)
% Derived quantities:  N (number of rows in allanarray), tvals[N],
% gyrox[N], gyroy[N], gyroz[N]
% Outputs: allandevx, allandevy, allandevz -- all are functions of tau

clear all;

%% Read in allanarray from file
allanarray = xlsread('data_set_name.xlsx');

%% Optional random removal of rows from allanarray to simulate nonuniformly
spaced time samples
% A new array, sparseallanarray, is created which is a copy of allanarray
% except that every seventh row of allanarray is either kept or discarded
% according to the value of a binary random variable r.

i = 1; % index for sparse array elements
j = 1; % index for allanarray elements
allanarraynumrows = size(allanarray,1);
sparseallanarray = zeros(allanarraynumrows,4);  % preallocation

while j < allanarraynumrows - 5
    sparseallanarray(i:i+5,:) = allanarray(j:j+5,:); % copy six rows
    r = randi(2,1);   % Returns binary random variable 1 or 2 with equal
probability
    if (r==2)
        sparseallanarray(i+6,:) = allanarray(j+6,:);  % Copy the seventh row
        i=i+7;
    else i=i+6; % Here, r=1. Don't copy the seventh row
    end
    j = j+7; % Move ahead to the next block of elements of allanarray
end

% copy the mod(allanarraynumrows,7)elements at the end of allanarray
remainder = mod(allanarraynumrows,7); % number of rows of allanarray than have
not yet been copied to sparseallanarray
nonzeroindices = find(sparseallanarray(:,1));  % Find nonzero elements in column
1, which is the time values
lastnonzeroelement = nonzeroindices(end);  % last nonzero element of time values
if ~isequal(remainder,0)
    sparseallanarray(lastnonzeroelement+1:lastnonzeroelement+remainder,:) =
allanarray(allanarraynumrows-remainder+1:allanarraynumrows,:);
end

% get rid of the extra zeroes at the end of sparseallanarray
sparseallanarray2 = sparseallanarray(1:lastnonzeroelement+remainder,:);
allanarray = sparseallanarray2;  %copy the sparse allan array into allanarray for
further processing


%% Break up big matrix into smaller matrices
tvals = allanarray(: , 1);
gyrox = allanarray(: , 2);
gyroy = allanarray(: , 3);
```

```
gyroz = allanarray(: , 4);
N = length (tvals);

%% Determine the range of valid tau values
% based on examination of tvals values. Tau cannot be too small or else
% there will not be enough elements to do proper bin averages. Tau cannot
% be too large or else there will not be enough bins to average over to
% calculate the Allan deviation. To calculate the minimum tau value,
% check out all windows of length 9 (the minimum number of elements needed
% to do a proper average) and tau must be at least as large as the window
% with the largest time span.

mask = ones(1, 9);
deltats = diff(tvals); % vector of differences of adjacent time values
w = conv(deltats, mask, 'valid');
taumin = max(w);
duration = tvals(N);
taumax = duration / 9.001;

%% Set up outer loop
% Calculate Allan variance for tau = taumin to taumax.
tauvals = logspace (log10(taumin), log10(taumax), 250);
numtaus = length (tauvals);

% Preallocation
allandevx = zeros(1, numtaus);
allandevy = zeros(1, numtaus);
allandevz = zeros(1, numtaus);
binnos = zeros(1, N);

%% Outer loop
for k = 1:numtaus
    tau = tauvals(k);



% Create startindices and endindices arrays based on the array of
% measurement times. Calculate endindices, and then startindices can easily
% be derived from it. binnos is an auxiliary vector whose components are
% the bin number of the corresponding element of tvals.
    startindices = zeros(1,N);   %preallocation
    endindices = zeros(1,N);     %preallocation
    binnos(1) = 1;
    startindices (1) = 1;

    % j is row number in tvals, and i is bin number.
    for j = 2:N
        if (isequal (mod (tvals(j) , tau), 0))
            binnos(j)  = uint32( tvals(j) / tau );
        else binnos(j) = uint32 ( tvals(j) / tau) + 1;
        end

        if ( binnos(j) > binnos(j-1))
            i = binnos(j-1);
            endindices(i) = j-1;
            startindices(i+1) = endindices(i)+1;
        end
    end
    endindices(binnos(N))= N;
    % If the final bin has fewer than 9 elements, exclude it from future
    % processing.
    if (endindices(binnos(N)) - startindices(binnos(N)) < 8)
        numbins = binnos(N) - 1;
    else numbins = binnos(N);
    end

% Calculate averages, deltas, squareds, sumofsquares, and allandev vectors
% Sum over each bin to get the bin averages

% Preallocation
avgsx = zeros(1, numbins);
avgsy = zeros(1, numbins);
avgsz = zeros(1, numbins);

   for i = 1:numbins
       si = startindices(i);
       ei = endindices(i);

       avgsx(i) = mean(gyrox(si:ei));
       avgsy(i) = mean(gyroy(si:ei));
       avgsz(i) = mean(gyroz(si:ei));
   end


   squaredsx = (diff(avgsx)).^2;
   squaredsy = (diff(avgsy)).^2;
   squaredsz = (diff(avgsz)).^2;

   allandevx(k) = (  sum(squaredsx) / (2 * (numbins -1))  )^0.5;
   allandevy(k) = (  sum(squaredsy) / (2 * (numbins -1))  )^0.5;
   allandevz(k)  = (  sum(squaredsz) / (2 * (numbins -1))  )^0.5;

end

%% Write the Allan deviations vectors into Excel files
% for later viewing and processing.  The units of these vectors are
% deg/sec.
xlswrite ('tauvals.xls', tauvals);
xlswrite ('allandevx.xls', allandevx);
xlswrite ('allandevy.xls', allandevy);
xlswrite ('allandevz.xls', allandevz);

%% Display the vectors in a log-log plot for immediate viewing.
% Convert deg/sec vectors to deg/hour.
allandevx = allandevx * 3600;
allandevy = allandevy * 3600;
allandevz = allandevz * 3600;

loglog (tauvals, allandevx, '-r', tauvals, allandevy, '-g', tauvals, allandevz,
'-b')
grid on
xlabel ('Time (seconds)');
ylabel ('Allan Deviation (degrees / hour)');
```

```
title('Allan Deviation for Gyro Data');
legend('X-Axis', 'Y-Axis', 'Z-Axis');
```

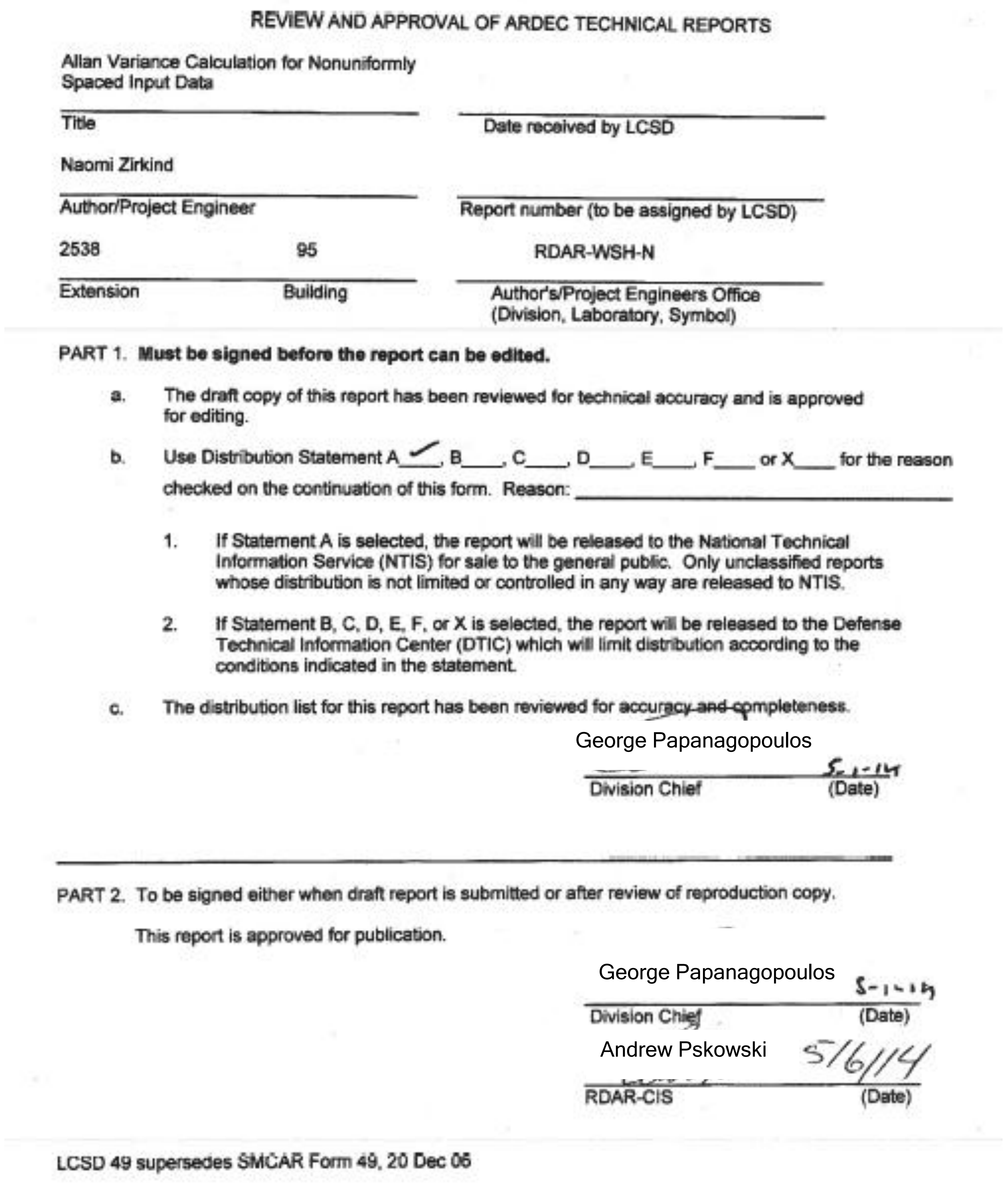
DISTRIBUTION LIST

U.S. Army ARDEC
ATTN: RDAR-EIK
RDAR-GC
RDAR-WSH-N, K. Patel
Picatinny Arsenal, NJ 07806-5000

Defense Technical Information Center (DTIC)
ATTN: Accessions Division
8725 John J. Kingman Road, Ste 0944
Fort Belvoir, VA 22060-6218

GIDEP Operations Center
P.O. Box 8000
Corona, CA 91718-8000
gidep@gidep.org

RDECOM CERDEC I2WD
ATTN: Naomi Zirkind
Room A3-120B
6003 Combat Drive
Aberdeen Proving Ground, MD 21005



REVIEW AND APPROVAL OF ARDEC TECHNICAL REPORTS

Allan Variance Calculation for Nonuniformly Spaced Input Data	
Title	Date received by LCSD
Naomi Zirkind	
Author/Project Engineer	Report number (to be assigned by LCSD)
2538 95	RDAR-WSH-N
Extension Building	Author's/Project Engineers Office (Division, Laboratory, Symbol)

PART 1. **Must be signed before the report can be edited.**

a. The draft copy of this report has been reviewed for technical accuracy and is approved for editing.

b. Use Distribution Statement A ✓, B____, C____, D____, E____, F____ or X____ for the reason checked on the continuation of this form. Reason: ____________________

1. If Statement A is selected, the report will be released to the National Technical Information Service (NTIS) for sale to the general public. Only unclassified reports whose distribution is not limited or controlled in any way are released to NTIS.

2. If Statement B, C, D, E, F, or X is selected, the report will be released to the Defense Technical Information Center (DTIC) which will limit distribution according to the conditions indicated in the statement.

c. The distribution list for this report has been reviewed for accuracy and completeness.

George Papanagopoulos   5-1-14
Division Chief   (Date)

PART 2. To be signed either when draft report is submitted or after review of reproduction copy.

This report is approved for publication.

George Papanagopoulos   5-1-14
Division Chief   (Date)

Andrew Pskowski   5/6/14
RDAR-CIS   (Date)

LCSD 49 supersedes SMCAR Form 49, 20 Dec 06